EPFL

# The SCARE-Frontier

## Practical Side-Channel-Assisted Reverse-Engineering Attack on Protected AES Ciphers

**Gaiëtan Renault**

School of Computer and Communication Sciences

MSc Semester Project

January 2023

**Responsible**
Prof. Serge Vaudenay
EPFL / LASEC

**Supervisor**
Andrea Caforio
EPFL / LASEC

LASEC

# Abstract

Side-Channel Assisted Reverse-Engineering (SCARE) is an attack technique aiming at recovering an undisclosed design of a cryptographic scheme from side-channel leakages of its implementation. First use of such technique in scientific literature was done by Novak in 2003 to attack the S-box design used in GSM protocol, by measuring consumed power of a smartcard [Nov03]. On another side, the Side-Channel-Assisted Differential Plaintext Attack (SCADPA) introduced by Breier [BJB18] and extended to See-In-The-Middle (SITM) attack by Bhasin [BBH+19] are effective methods to recover the secret key of a cryptosystem with a Substitution-Permutation Network (SPN) structure. SCADPA is a side-channel power analysis attack that exploits leakages in differential power traces induced by plaintext differentials. SITM attack is a SCADPA strategy that targets partially masked SPN implementations. Caforio et al. [CBB21] successfully deployed this SCADPA attack to recover the hidden component functions of an unprotected Advanced Encryption Standard (AES) custom cipher.

In practice, software implementation of AES-like ciphers tends to be enhanced with countermeasures, like instructions shuffling or S-box masking, to thwart side-channel attacks [RPD09], [AG]. Such protective measures were for example adopted by the ANSSI standardization agency [LoES18].

Below work demonstrate how the SCAPDA technique can be applied to practically recover the complete description of a protected AES cipher, implementing instructions shuffling at SB and AK layers, with a 128-bit key size. As a first step, we review the practical SCAPDA approach from Caforio et al. paper. Then, as a second step, this methodology is extended to the protected AES implemented on a 32-bit architecture microcontroller.

# Contents

# 1 Introduction

This project was carried out as part of the COM-416 - Semester project in communication systems II course from August 2022 to January 2023. This work was supervised throughout by PhD student Andrea Caforio whom I sincerely thank for his precious and continuous help. This project has been done under the responsibility of Professor Serge Vaudenay and as part of the Security and Cryptography Laboratory (LASEC) at EPFL.

## 1.1 Background : An Overview of the Side-Channel Attacks

In cryptography, while AES is generally considered to be a secure symmetric encryption algorithm, it is not immune to attacks. Side-channels are highly effective attack vectors exploiting the physical implementation of a system. Any cryptographic algorithm implementation physically interact with his environment. A Side-Channel Attack (SCA) could target and evaluate those interactions such as power consumption, elapsed time or electromagnetic noise to gain valuable information in the cryptanalysis process. One of the earliest use of SCA to cryptographic operations was the DPA attack, described by P. Kocher et al. in 1998 [KJJ99]. This attack was based on current and power consumption measurements and was targeting the secret keys values in DES. DPA make use of the observation that different cryptographic operations performed on a device will result in different measured patterns exhibited by the power consumption. DPA observes and analyzes the differential between two power traces, each one corresponding to a power measurement during the encryption process of a different chosen-plaintext. In overall, DPA attacks can be difficult to defend against and can be performed in a simple and inexpensive way.

A more refined form of DPA, termed SCADPA, was proposed in 2018 by Breier et al. [BJB18] and was conducted on the PRESENT cipher. The attack targets the first round of the S-box layer and observes the power trace amplitude differential. Each trace is resulting from the encryption of two different chosen-plaintexts with one bit difference at position $i$. By taking several such differentials one can recover one key bit at position $i$. SCADPA attack is applicable to ciphers having as diffusion function only bit permutation. SCADPA was extended by Bhasin et al. to the See-In-The-Middle (SITM) attack [BBH$^+$19] to be applicable on partially masked AES where the middle rounds remain unprotected and where the middle or corner rounds implement instructions shuffling.

On another hand, in a SCARE attack, the information leakage is not only use to recover the cryptographic secrets but also to reverse-engineer some hidden structure in the system, typically the components functions of AES-like ciphers (like hidden SubBytes, ShiftRows and MixColumns functions). In 2013, the first publications mention the SCARE attack applied on AES-like ciphers [CIW13], [RR13] where some components are kept secret. Those two works are theoretical analysis of the SCARE attack and bring out the capabilities of SCARE methodology to recover any hidden cryptographic structure with relatively low complexity. We have to wait 2020 to see the first practical attack on a hidden S-box function [JB20] and 2021 for the first practical attack on the complete hidden cryptographic structure of AES-like cipher [CBB21].

In order to defend against side-channel attacks AES-like ciphers tend to be implemented with defensive mechanisms. The protection considered in this work is instructions shuffling of the AK and SB layers. This frequently used type of countermeasure was first introduced by [HOM06], [RPD09] and Amarilli et al [AMN$^+$11] suggested it as good security improvement. Since then, Veyrat-Charvillon et al. [VCMKS12] conducted an information theoretic analysis mitigating this strong security claim for simplified versions of shuffling (i.e instruction partial permutations starting from a Random Start Index called RSI) and highlighted presence of indirect and direct leakages during the shuffling step. In this work, an orthogonal approach is taken as there is no use of any side leakage model of the shuffling

operation.

## 1.2 Contributions

All the code of this project is available on GitHub repository [Ren22]. Below are the contributions of this project.

- In the first part we verify and review the SCARE paper from Caforio et al. by re-implementing the full pipeline attack to reverse-engineer the component functions of the AES-128 scheme.
- We show an inexpensive and accurate profiling technique able to recover the Hamming weight after the AK and SB layers.
- Based on this newly introduced profiling technique we create an inexpensive SCARE attack on protected SPN implementing instructions shuffling at the AK and SB layers. Thanks to the Caforio et al. method, the presented SCARE considers the instructions shuffling steps as black-box operations with no visible leakage and still recovers at the end the AK, SB, PB and MC full description.

## 1.3 Outline

The first section is a close review of the work done by Caforio et al. [CBB21] presenting practical SCARE approach to recover a full description of unprotected AES-128 cipher implementing undisclosed AddRoundKey (AK), SubBytes (SB), ShiftRows (PB) and Mix-Columns (MC) functions. The key ingredients of above approach were the design of an algorithm computing the Hamming weight of the state bytes after the AK function and the use of DPA to compute the differential state bytes activeness. The second part is presenting a new method based on above methods to attack and recover a full description of protected AES-128 cipher deploying the same functions.

## 1.4 Adversary Model

Here is a presentation of common assumed adversary model in a SCA scenario. The attack is conducted, in a gray-box model, on the software implementation of a microcontroller. The attacker has access to basic details like the algorithm being used (i.e. AES) and its structure (e.g. number of rounds, SPN), but does not have complete knowledge of the system. In particular the attacker is not aware of the key, S-box, MixColumns and Permutation details. The attacker can submit plaintexts for encryption using a fixed, unknown key and observes the resulting ciphertexts. The adversary exploits side-channel information, specifically it can detect changes in the intermediate state bytes for two different encryptions.

## 1.5 Notation

All operations related to AES scheme, are done in the Rijndael-Galois Field (GF), denoted by $\mathbb{F}_{256}$.

- $x \mapsto HW(x)$ : Hamming weight of the byte x. For example $HW(106) = 4$.
- An empty-initialized array is written $L = []$. Arrays follow the zero-based array indexing approach. The access of the $i$-th element of an array follows square-bracket notation $L[i]$. Slicing follows $L[start : stop]$ convention, where *start* is inclusive and *stop* exclusive.

- Let $L = [x_0, ..., x_n]$ be an array. Then argsort $L$ is the array $[0, ..., n]$ sorted by the order defined by $i < j$ if $L[i] < L[j]$ or $L[i] = L[j]$ and $i < j$ in the usual order of integers.
- An empty-initialized set is written $L = \{\}$.
- The cardinality of a set $S$ is denoted by $|S|$.
- $S_n$ is the group permutation of all permutations of $n$ elements.
- We call state bytes a $\mathbb{F}_{256}^{4\times4}$ column-major order matrix of 16 bytes. The columns are numbered from zero to three.
- The input plaintext state bytes is denoted:

$$\begin{bmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix}$$

- $KS$: The key scheduling (or expansion) layer. The key expansion is performed once at beginning of the algorithm by a key scheduling procedure. This layer takes the 128-bit encryption key and expands it into several round keys that are used in the subsequent rounds of the AES encryption process. This layer will not be studied.
- $AK(i)$: The AddRoundKey layer at round $i \in \{0, ..., 10\}$, where $i = 0$ is initial round. This layer is described by the following AddRoundKey procedure where $k$ a round key and $s$ a state bytes. The $AK$ layer implementation follows the column-major order.

**Algorithm 1:** AddRoundKey

```
Input: k, s
Output: s
for i ← 0 to 3 do
    for j ← 0 to 3 do
        s[i, j] ← s[i, j] ⊕ k[i, j]
```

- $SB(i)$: The SubBytes layer at round $i \in \{1, ..., 10\}$. This layer is described by the following SubBytes procedure where $T$ is a substitution and $s$ a state bytes. The $SB$ layer implementation follows the row-major order.

**Algorithm 2:** SubBytes

```
Input: T, s
Output: s
for i ← 0 to 3 do
    for j ← 0 to 3 do
        s[j, i] ← T(s[j, i])
```

- $PB(i)$: The Permutation bytes layer at round $i \in \{1, ..., 10\}$. This layer is described by the following Permutation procedure where $\Pi$ is a permutation and $s$ a state bytes. This layer is not targeted by side-channel leakages.

**Algorithm 3:** Permutation

**Input:** $\Pi$, $s$
**Output:** $s$
**for** $i \leftarrow 0$ **to** 3 **do**
  **for** $j \leftarrow 0$ **to** 3 **do**
    $s_{\text{copy}}[i,j] \leftarrow s[i,j]$
**for** $i \leftarrow 0$ **to** 3 **do**
  **for** $j \leftarrow 0$ **to** 3 **do**
    $s[i,j] \leftarrow s_{\text{copy}}[\Pi(i,j)]$

- $MC(i)$: MixColumns layer at round $i \in \{1, ..., 10\}$. This layer is described by the following MixColumns procedure where $M$ is a $4 \times 4$ circulant matrix and $s$ a state bytes. We consider $s[i]$ as a column vector. This layer is not targeted by side-channel leakages.

**Algorithm 4:** MixColumns

**Input:** $M$, $s$
**Output:** $s$
**for** $i \leftarrow 0$ **to** 3 **do**
  $s[i] \leftarrow M \times s[i]$

- The state bytes matrix after layer $A \in \{AK, SB, PB, MC\}$ is :

$$\begin{bmatrix} b_{0,A} & b_{4,A} & b_{8,A} & b_{12,A} \\ b_{1,A} & b_{5,A} & b_{9,A} & b_{13,A} \\ b_{2,A} & b_{6,A} & b_{10,A} & b_{14,A} \\ b_{3,A} & b_{7,A} & b_{11,A} & b_{15,A} \end{bmatrix}$$

  For example $b_{11,SB(1)}$ is the 11th state bytes output after the $SB$ function at first round.

- Used AES-128 algorithm is described by Algorithm 5.

**Algorithm 5:** AES-128

$k \leftarrow KS(k)$
$AK(0; k^0, s)$
**for** $i \leftarrow 1$ **to** 9 **do**
  $SB(i; T, s)$, $PB(i; \Pi, s)$, $MC(i; M, s)$, $AK(i; k^i, s)$
$SB(i; T, s)$, $PB(i; \Pi, s)$, $AK(i; k^{10}, s)$

# 2 SCARE Preliminaries

## 2.1 Setup

The Device Under Test (DUT) is the 32-bit microcontroller STM32F415, embedding the Cortex™-M4 core frequenced at 168 MHz, and connected to the CW308 UFO target board solution from NewAE Technology Inc. Power measurement is performed by the

```
; Round key addition
LD R1, [ADDR PT]
LD R2, [ADDR KEY]
XOR R1, R2
ST R1, [ADDR PT]
```

```
; Byte substitution
LD R1, [ADDR STATE]
ADD R2, R1, [ADDR SBOX]
LD R3, R2
ST R3, [ADDR STATE]
```

**Figure 1:** Byte-wise assembly instructions of the AddRoundKey function on the left side and of the SubBytes function on the right side. This assembly instructions figure comes from Caforio et al. paper[CBB21].

ChipWhisperer-Lite 32-Bit. Implemented AES scheme is inspired from the straightforward byte-wise implementation *tiny-AES-128.C* [Inc22].

Below is an instructions description of each function.

- AK: The non-shuffled AK function is defined as the ordered byte-wise xor operation between the state bytes matrix and the round keys. This round key addition operation is described at assembly level in Figure 1. In the shuffled AK function the xor instructions are not necessarily done in the state bytes order but follow a 16 (to 16) random permutation implemented via a look-up table. This random table is computed using Fisher–Yates shuffle algorithm via the C pseudo-random generator function `rand()`. We denote by $\Pi^{(k)}$ this associated AK random instructions permutation at initial round. Equivalently, $\Pi^{(k)}(i) = j$ is such that the $j$-th instruction is executed at $i$-th position.

- SB: The non-shuffled SB function is defined as the direct addressing of the state bytes using any hidden and fixed $8 \times 8$ S-box look-up table denoted by $T$. This bytes substitution operation is described at the assembly level in Figure 1. The order of the direct addressing is a row-major order (opposed to the column-major state bytes order). The instructions shuffled version of it perform the direct addressing in a random order using a look-up table generated in the same way as the one used in shuffled AK function. We denote by $\Pi^{(s)}$ this associated SB random instructions permutation at first round. Equivalently, $\Pi^{(s)}(i) = j$ is such that the $j$-th instruction is executed at $i$-th position.

- PB: The PB function is defined as a fixed 16 to 16 random permutation, denoted by $\Pi \in S_{16}$, applied to the state bytes. Equivalently, $\Pi(i) = j$ is such that the $j$-th byte is mapped to $i$-th position. This is implemented using a random look-up table, created via a similar approach compared to the shuffled AK look-up table.

- MC: The MC function is defined as a linear diffusion layer multiplying the state bytes matrix by a circulant matrix $M$.

$$M := \begin{pmatrix} a & b & c & d \\ d & a & b & c \\ c & d & a & b \\ b & c & d & a \end{pmatrix}$$

  where $a, b, c, d \in \mathbb{F}_{256}^*$.

- $KS$: The key scheduling function is defined as the 10 rounds AES key schedule where S-box function is replaced by the $T$ function.

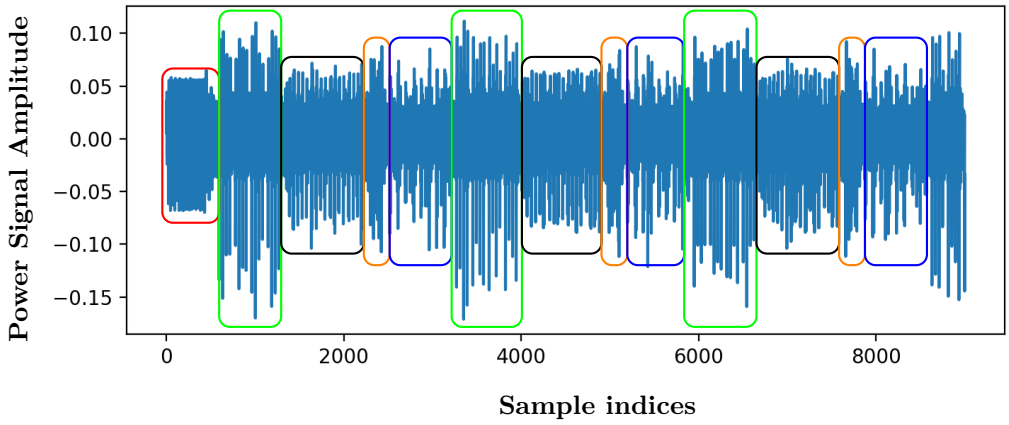


**Figure 2:** Average power trace of the first three rounds. In red, green, black, orange, blue the $KS, AK, SB, PB, MC$ layers respectively. Trace corresponding to $AK$ or $SB$ layers are composed of 16 power spikes, each spike is associated with a memory-type instruction and is correlated with the Hamming weight of the manipulated state bytes.

## 2.2 Power Trace Capture

**Definition (Power Trace).** We take back a similar definition of power trace than the one used by Caforio et al. More precisely we define the power trace $E(b_{i,A(j),p})$ to be the power signal measured during computation of byte $b_{i,A(j)}$ during encryption of plaintext $p$, where $i \in \{0, ..., 15\}$, $j \in \{0, ..., 10\}$ and $A \in \{AK, SB, PB, MC\}$. We will sometimes omit $p$ when the plaintext is irrelevant. $\bar{E}(b_{i,A(j)})$ is defined as the averaged power signal over several measurements. We typically take the average over $\alpha$ measurements, where $\alpha := 10$ was found as the number of power traces required to empirically observe a stable power trace average.

Power traces are floating-point values mapped between -0.5 and 0.5 with a 10-bit Analog to Digital Converter (ADC) resolution. As an example, Figure 2 is a power trace measurement of the first three AES encryption rounds.

In SCA, the well known Hamming-weight model [MOP07] tells us that $\bar{E}(b_{i,AK(j),p}) > \bar{E}(b_{i,AK(j),p'})$ if and only if $HW(b_{i,AK(j),p}) > HW(b_{i,AK(j),p'})$ where $p$ and $p'$ are distinct plaintexts. It also applies to SB layer; $\bar{E}(b_{i,SB(j),p}) > \bar{E}(b_{i,SB(j),p'})$ if and only if $HW(b_{i,SB(j),p}) > HW(b_{i,SB(j),p'})$.

## 2.3 Hidden AES layers

### 2.3.1 AddRoundKey Layer

We denote by $K_i = (k_0^i, ..., k_{15}^i)$ the round key of the $i$-th round. We have for the AK layer at initial round :

$$\begin{bmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix} \xrightarrow{\text{AK}(0)} \begin{bmatrix} b_{0,AK(0)} & b_{4,AK(0)} & b_{8,AK(0)} & b_{12,AK(0)} \\ b_{1,AK(0)} & b_{5,AK(0)} & b_{9,AK(0)} & b_{13,AK(0)} \\ b_{2,AK(0)} & b_{6,AK(0)} & b_{10,AK(0)} & b_{14,AK(0)} \\ b_{3,AK(0)} & b_{7,AK(0)} & b_{11,AK(0)} & b_{15,AK(0)} \end{bmatrix}$$

$$= \begin{bmatrix} p_0 + k_0^0 & p_4 + k_4^0 & p_8 + k_8^0 & p_{12} + k_{12}^0 \\ p_1 + k_1^0 & p_5 + k_5^0 & p_9 + k_9^0 & p_{13} + k_{13}^0 \\ p_2 + k_2^0 & p_6 + k_6^0 & p_{10} + k_{10}^0 & p_{14} + k_{14}^0 \\ p_3 + k_3^0 & p_7 + k_7^0 & p_{11} + k_{11}^0 & p_{15} + k_{15}^0 \end{bmatrix}$$

In a non shuffling setting of AK layer we have that $E(b_{i,AK(0)})$ is the power trace capture corresponding to encryption of byte $b_i$ at layer $AK(0)$.

### 2.3.2 SubBytes Layer

We have for the Substitution Bytes (SB) layer at first round :

$$\begin{bmatrix} b_{0,AK(0)} & b_{4,AK(0)} & b_{8,AK(0)} & b_{12,AK(0)} \\ b_{1,AK(0)} & b_{5,AK(0)} & b_{9,AK(0)} & b_{13,AK(0)} \\ b_{2,AK(0)} & b_{6,AK(0)} & b_{10,AK(0)} & b_{14,AK(0)} \\ b_{3,AK(0)} & b_{7,AK(0)} & b_{11,AK(0)} & b_{15,AK(0)} \end{bmatrix} \xrightarrow{\text{SB(1)}} \begin{bmatrix} b_{0,SB(1)} & b_{4,SB(1)} & b_{8,SB(1)} & b_{12,SB(1)} \\ b_{1,SB(1)} & b_{5,SB(1)} & b_{9,SB(1)} & b_{13,SB(1)} \\ b_{2,SB(1)} & b_{6,SB(1)} & b_{10,SB(1)} & b_{14,SB(1)} \\ b_{3,SB(1)} & b_{7,SB(1)} & b_{11,SB(1)} & b_{15,SB(1)} \end{bmatrix}$$

$$= \begin{bmatrix} T(b_{0,AK(0)}) & T(b_{4,AK(0)}) & T(b_{8,AK(0)}) & T(b_{12,AK(0)}) \\ T(b_{1,AK(0)}) & T(b_{5,AK(0)}) & T(b_{9,AK(0)}) & T(b_{13,AK(0)}) \\ T(b_{2,AK(0)}) & T(b_{6,AK(0)}) & T(b_{10,AK(0)}) & T(b_{14,AK(0)}) \\ T(b_{3,AK(0)}) & T(b_{7,AK(0)}) & T(b_{11,AK(0)}) & T(b_{15,AK(0)}) \end{bmatrix}$$

where the SB fixed lookup table $T : \mathbb{F}_{256} \to \mathbb{F}_{256}$ is unknown to the adversary.

### 2.3.3 Permutation Layer

$$\begin{bmatrix} b_{0,SB(1)} & b_{4,SB(1)} & b_{8,SB(1)} & b_{12,SB(1)} \\ b_{1,SB(1)} & b_{5,SB(1)} & b_{9,SB(1)} & b_{13,SB(1)} \\ b_{2,SB(1)} & b_{6,SB(1)} & b_{10,SB(1)} & b_{14,SB(1)} \\ b_{3,SB(1)} & b_{7,SB(1)} & b_{11,SB(1)} & b_{15,SB(1)} \end{bmatrix} \xrightarrow{\text{PB(1)}} \begin{bmatrix} b_{0,PB(1)} & b_{4,PB(1)} & b_{8,PB(1)} & b_{12,PB(1)} \\ b_{1,PB(1)} & b_{5,PB(1)} & b_{9,PB(1)} & b_{13,PB(1)} \\ b_{2,PB(1)} & b_{6,PB(1)} & b_{10,PB(1)} & b_{14,PB(1)} \\ b_{3,PB(1)} & b_{7,PB(1)} & b_{11,PB(1)} & b_{15,PB(1)} \end{bmatrix}$$

$$= \begin{bmatrix} b_{\Pi(0),SB(1)} & b_{\Pi(4),SB(1)} & b_{\Pi(8),SB(1)} & b_{\Pi(12),SB(1)} \\ b_{\Pi(1),SB(1)} & b_{\Pi(5),SB(1)} & b_{\Pi(9),SB(1)} & b_{\Pi(13),SB(1)} \\ b_{\Pi(2),SB(1)} & b_{\Pi(6),SB(1)} & b_{\Pi(10),SB(1)} & b_{\Pi(14),SB(1)} \\ b_{\Pi(3),SB(1)} & b_{\Pi(7),SB(1)} & b_{\Pi(11),SB(1)} & b_{\Pi(15),SB(1)} \end{bmatrix}$$

where PB is described by a random and fixed permutation $\Pi : \{0, ..., 15\} \to \{0, ..., 15\}$ unknown to the adversary.

### 2.3.4 MixColumns Layer

$$\begin{bmatrix} b_{0,PB(1)} & b_{4,PB(1)} & b_{8,PB(1)} & b_{12,PB(1)} \\ b_{1,PB(1)} & b_{5,PB(1)} & b_{9,PB(1)} & b_{13,PB(1)} \\ b_{2,PB(1)} & b_{6,PB(1)} & b_{10,PB(1)} & b_{14,PB(1)} \\ b_{3,PB(1)} & b_{7,PB(1)} & b_{11,PB(1)} & b_{15,PB(1)} \end{bmatrix} \xrightarrow{\text{MC(1)}} \begin{bmatrix} b_{0,MC(1)} & b_{4,MC(1)} & b_{8,MC(1)} & b_{12,MC(1)} \\ b_{1,MC(1)} & b_{5,MC(1)} & b_{9,MC(1)} & b_{13,MC(1)} \\ b_{2,MC(1)} & b_{6,MC(1)} & b_{10,MC(1)} & b_{14,MC(1)} \\ b_{3,MC(1)} & b_{7,MC(1)} & b_{11,MC(1)} & b_{15,MC(1)} \end{bmatrix}$$

$$= \begin{bmatrix} M \times \begin{pmatrix} b_{0,MC(1)} \\ b_{1,MC(1)} \\ b_{2,MC(1)} \\ b_{3,MC(1)} \end{pmatrix} & M \times \begin{pmatrix} b_{4,MC(1)} \\ b_{5,MC(1)} \\ b_{6,MC(1)} \\ b_{7,MC(1)} \end{pmatrix} & M \times \begin{pmatrix} b_{8,MC(1)} \\ b_{9,MC(1)} \\ b_{10,MC(1)} \\ b_{11,MC(1)} \end{pmatrix} & M \times \begin{pmatrix} b_{12,MC(1)} \\ b_{13,MC(1)} \\ b_{14,MC(1)} \\ b_{15,MC(1)} \end{pmatrix} \end{bmatrix}$$

where the MC is described by a fixed and random circulant matrix $M$ unknown to the adversary.

## 2.4 Protected AES Layers

### 2.4.1 Masking layers

Masking is a commonly used security mechanism against side-channel attack that could be proved to be resistant to first-order differential side-channel attack [KHL11]. First-order DPA refers to an attack having access to only one power measurement source during the side-channel leakage of a differential input. In our given threat model, the adversary is thus unable to succeed the attack as it has only control over one input, the submitted plaintext, and over one measured leakage, the power trace measurement.

Masking consists of performing some operations (like addition or multiplication) with random mask values at intermediate steps to obscure the input-output relationship. For example one common way to do it is to mask a secret variable by randomly splitting it into $s + 1$ shares, where $s$ is order of the masking. Higher-order masking can achieve any desired resistance level by trading efficiency for security [KHL11]. Non-linear layers need extra care when it comes to masking as described by Canright et al. [CB08].

### 2.4.2 Instructions Shuffling

An other commonly used defensive mechanism is instructions shuffling, it is the one studied in details in this project. Instructions shuffling is typically applied to only a part of the instructions executed by the system, as uniform random shuffling comes with performance overheads. The optimal shuffling algorithm is the Fisher–Yates shuffle that can be run in-place and has a linear time complexity with the number of instructions to shuffle.

Algorithm 6 illustrates the used protected AES implementing instructions shuffling at AK and SB layers via the Fisher–Yates shuffle algorithm.

**Algorithm 6:** Protected AES-128

$k \leftarrow KS(k)$ Fisher_Yates(AK instructions) `// Shuffle the AK instructions using Fisher-Yates shuffle algorithm`
$AK(0; k, s)$
**for** $i \leftarrow 1$ **to** 9 **do**
    Fisher_Yates(SB instructions),
    $SB(i; T, s)$, $PB(i; \Pi, s)$, $MC(i; M, s)$, Fisher_Yates(SB instructions),
    $AK(i; k^i, s)$
Fisher_Yates(SB instructions),
$SB(i; T, s)$,
$PB(i; \Pi, s)$,
Fisher_Yates(SB instructions),
$AK(i; k^{10}, s)$

Presence of the shuffling algorithm can be located in the power trace but will not be used in the work as it is assumed to be black-boxed. Figure 3 represents respectively, a unique (above) and an average over 100 runs (bottom) power trace of the AK layer. There are very large differences in their power trace amplitude. This is naturally explained by the instructions shuffling protection; the average is done over 100 random shuffles, and amplitude peaks are thus mixed. Average power trace is not leaking information and is not practical anymore.

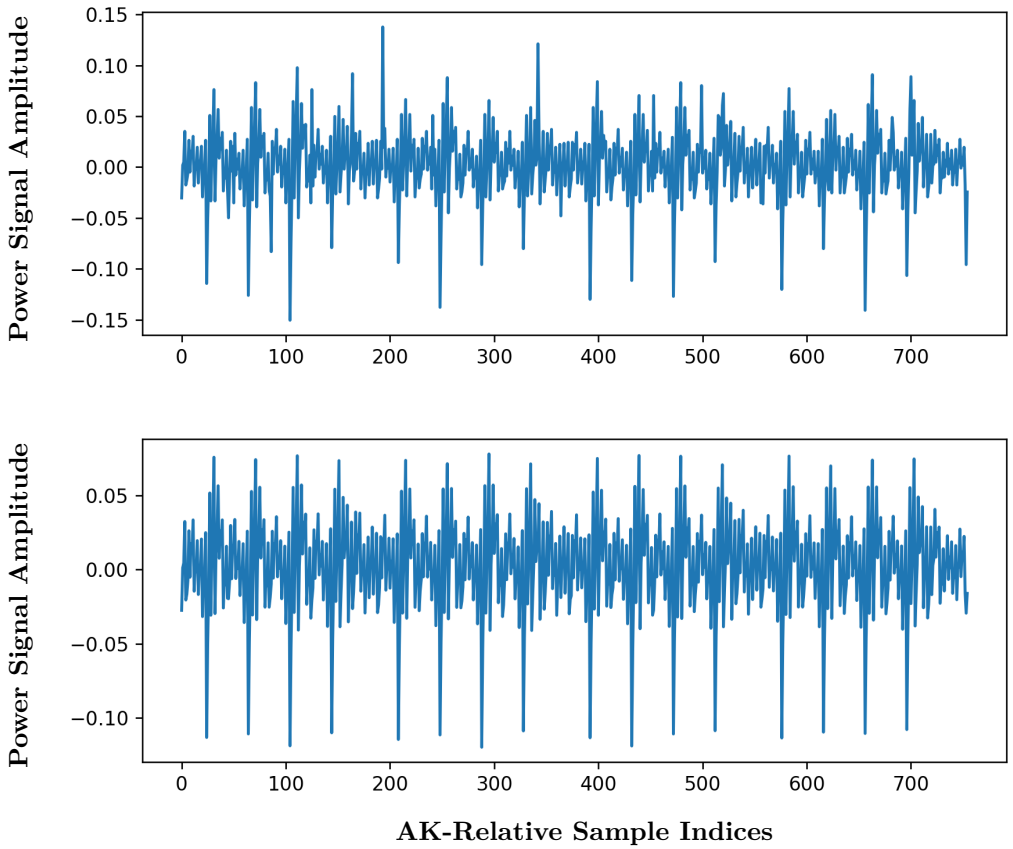


**Figure 3:** Top plot is one power trace capture of the AK layer implementing instructions shuffling. For each new power trace capture we will observe a new unstable trace with randomly shuffled spikes. Bottom plot is an average power trace of AK layer implementing instructions shuffling. The spikes are similar as they are each an average spike induced over all $AK$ state bytes.

# 3 Reviewing practical SCADPA methodology for AES-Like Ciphers

This section is a review of the SCADPA attack methodology for AES-like cipher presented by Caforio et al. [CBB21].

## 3.1 Attack on the AddRoundKey Layer

First step of the attack is to recover the round keys obtained after the key expansion. Those keys fully describe the AK layer at initial round. The SCADPA observations from Section 2.2 justify Algorithm 7 recovering the Hamming weight $H(b)$ for $b \in \{b_{i,AK(0),p}, b_{i,SB(1),p}\}$.

One can recover the $i$-th round key by finding the byte $p'_i$ such that $b_{i,AK/SB(i)} = p'_i + k_i^0 = 0$. Exhaustively, Algorithm 8 shows how an adversary obtain full AK layer description.

As an important note, Algorithm 7 can be modified into Algorithm 9 to enforce the Hamming weight $H(b)$ for $b \in \{b_{i,AK(0),p}, b_{i,SB(1),p}\}$.

**Algorithm 7:** Recover the Hamming weight $HW(b)$ for $b \in \{b_{i,AK(0),p}, b_{i,SB(1),p}\}$

```
   // Initialize an array based on observation in Figure 4
1  L ← [255, 246, 210, 126, 0, −126, −210, −246, −255]
   // Given fixed plaintext p, compute average power trace
2  e ← Ē(b_{i,AK/SB(i),p})
   // h counts for all different plaintext bytes the difference between
      number of state bytes with greater HW and the number of state
      bytes with lower HW than b_{i,AK/SB(i),p}.
3  h ← 0, p ← p'
   // Iterate over all plaintext bytes t
4  for t ← F_256\{p_i} do
5  |  p'_i ← t
6  |  e' ← Ē(b_{i,AK/SB(i),p'}) // Compute power trace of p'
   |  // Update h accordingly
7  |  if e' < e then
8  |  |  h ← h − 1
9  |  else if e' > e then
10 |  |  h ← h + 1
   // Finds the HW of b_{i,AK/SB(i),p}
11 return argmin_{j∈{0,...,8}} |h − L[j]|
```

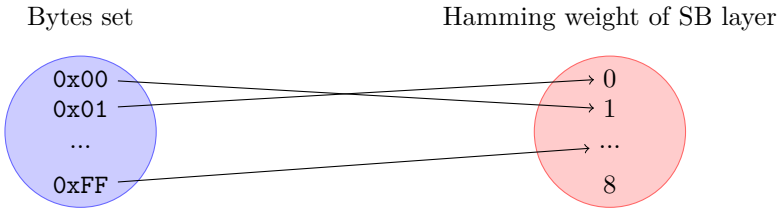


**Figure 4:** Mapping diagram of $b \mapsto HW(AK(b)) = HW(b + k)$ from bytes set $\mathbb{F}_{256}$ to Hamming weights set $\{0, ..., 8\}$ for any fixed byte $k$. As AK is a one-to-one function then $|\{b \in \mathbb{F}_{256}, HW(AK(b)) = i\}| = \binom{8}{i}$

**Complexity:** The parallelization trick, attacking all state bytes at same time, reduces the complexity cost down to $\alpha \times 2^8$, where $\alpha$ is the number of repetitions required to get a stable average power trace. Indeed, we observed spikes sufficiently spaced apart to conduct the parallelization trick over every plaintext bytes. More specifically, we iteratively set all the plaintext bytes to $j \in \mathbb{F}_{256}$ and deduce from each spike if associated the state byte after $AK$ has a null Hamming weight.

## 3.2 Attack on Permutation Layer

As a next target, the MC layer will be the second layer to be (partially) reverse-engineered. Using DPA, the attacker will be able to find $\Pi$ up to row permutation via differential activeness.

**Algorithm 8:** Recover the AddRoundKey

$p = p'$
**for** $l \leftarrow 0$ **to** 15 **do**
  L = [], K = []
  **for** $t \leftarrow \mathbb{F}_{256}$ **do**
    $p'_i \leftarrow t$
    $e' \leftarrow \bar{E}(b_i, AK/SB(i), p')$
    $L[t] \leftarrow e'$
  $K[l] \leftarrow \text{argmin } L$
**return** $K$

**Algorithm 9:** Set the Hamming weight $HW(b)$ to $h$ for $b \in \{b_{i,AK(0)}, b_{i,SB(1)}\}$

// $L$ is built such that if we order the HW of all 256 bytes then we have that the $L[ht]$-th byte has HW $h$
$L \leftarrow \{0:0, 1:8, 2:36, 3:92, 4:162, 5:218, 6:246, 7:254, 8:255\}$
$T \leftarrow []$ **for** $t \leftarrow \mathbb{F}_{256}$ **do**
  $p_i \leftarrow t$
  $e \leftarrow \bar{E}(b_{i,AK/SB(i)})$
  $T[t] \leftarrow e$
// Returns the byte value $b$ such that $HW(b) = h$ for $b \in \{b_{i,AK(0)}, b_{i,SB(1)}\}$
**return** (argsort $T$)$[L[ht]]$

**Definition (Differential activeness):** Caforio et al. define a byte to be differentially active, noted $\delta(b_{i,A(j)}) = 1$, if the encryption of two different plaintexts $p$ and $p'$ leads to $b_{i,A(j),p'} \neq b_{i,A(j),p}$.

The differential plaintext attack inserts a single plaintext byte difference at $i$-th position such that $p_i \neq p'_i$. For example during the first round, attacker can observe the differential average power traces at $AK$ or $SB$ layer placed after the $PB(1)$ layer (i.e. $|\bar{E}(b_{j,AK/SB(i),p}) - \bar{E}(b_{j,AK/SB(i),p'})| > \epsilon$ for a small positive $\epsilon$) to find the four indices $j$ in the same column as index $\Pi(i)$ since $\delta(b_{j,AK/SB(1)}) = 1$.

**Example:** As an example, let DPA changes the $p_0$ plaintext byte. Then at the $PB(1)$ layer the $b_{0,PB(1)}$ byte is moved to index position $\Pi(i)$ and the $MC(1)$ layer diffuses this differential activeness in the column of the byte index $\Pi(i)$. The equations below illustrates this situation.

The plaintexts are equal to :

$$p = \begin{bmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix}$$

$$p' = \begin{bmatrix} p'_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{bmatrix}, \text{ where } p'_0 \neq p_0$$

The intermediate differential state bytes matrices at the layer $AK(0)$ or $SB(1)$ are equal to :

$$M_A^\delta := \begin{bmatrix} \delta(b_{0,A}) & \delta(b_{4,A}) & \delta(b_{8,A}) & \delta(b_{12,A}) \\ \delta(b_{1,A}) & \delta(b_{5,A}) & \delta(b_{9,A}) & \delta(b_{13,A}) \\ \delta(b_{2,A}) & \delta(b_{6,A}) & \delta(b_{10,A}) & \delta(b_{14,A}) \\ \delta(b_{3,A}) & \delta(b_{7,A}) & \delta(b_{11,A}) & \delta(b_{15,A}) \end{bmatrix}, \text{ where } A \in \{AK(0), SB(1)\}$$
$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$PB(1)$ moves the zeroth differential active byte to the $\Pi(0)$ position. For illustration purposes let us assume that $\Pi(0) = 10$. The intermediate differential state byte matrix at the layer $PB(1)$ is then equal to :

$$M_{PB(1)}^\delta := \begin{bmatrix} \delta(b_{0,PB(1)}) & \delta(b_{4,PB(1)}) & \delta(b_{8,PB(1)}) & \delta(b_{12,PB(1)}) \\ \delta(b_{1,PB(1)}) & \delta(b_{5,PB(1)}) & \delta(b_{9,PB(1)}) & \delta(b_{13,PB(1)}) \\ \delta(b_{2,PB(1)}) & \delta(b_{6,PB(1)}) & \delta(b_{10,PB(1)}) & \delta(b_{14,PB(1)}) \\ \delta(b_{3,PB(1)}) & \delta(b_{7,PB(1)}) & \delta(b_{11,PB(1)}) & \delta(b_{15,PB(1)}) \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

The $MC(1)$ layer diffuses the tenth differential active byte (in the layer $PB(1)$) to the column of the tenth byte index (i.e. column 2).The intermediate differential state bytes matrices at the layer $MC(1), AK(1)$ or $SB(2)$ are then equal to :

$$M_A^\delta := \begin{bmatrix} \delta(b_{0,A}) & \delta(b_{4,A}) & \delta(b_{8,A}) & \delta(b_{12,A}) \\ \delta(b_{1,A}) & \delta(b_{5,A}) & \delta(b_{9,A}) & \delta(b_{13,A}) \\ \delta(b_{2,A}) & \delta(b_{6,A}) & \delta(b_{10,A}) & \delta(b_{14,A}) \\ \delta(b_{3,A}) & \delta(b_{7,A}) & \delta(b_{11,A}) & \delta(b_{15,A}) \end{bmatrix}, \text{ where } A \in \{MC(1), AK(1), SB(2)\}$$
$$= \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The Figure 5 illustrates the DPA attack to recover the bytes differential activeness and thus recover $\Pi$ up to row permutation. More specifically, a similar power trace spikes location to the blue one implies that the plaintexts differential activates the zeroth column, and the orange, green and red one the first, second and third column respectively. The attacker obtains a full description of the $\Pi$ layer in next section.

**Complexity:**: In order to compute $\Pi$ up to row permutations the attack requires $16 \times 2 \times \alpha$ power traces. Indeed, for each of the 16 state bytes, one need two average power traces to compute a differential power trace.

## 3.3 Attack on MixColumns Layer

After having partially recovered the $\Pi$ permutation, next stage is to find $MC$ matrix $M$ via filter equation solving. Let us denote $u_i \in \{0, 1, 2, 3\}$ be such that $\Pi(u_i) = i$,

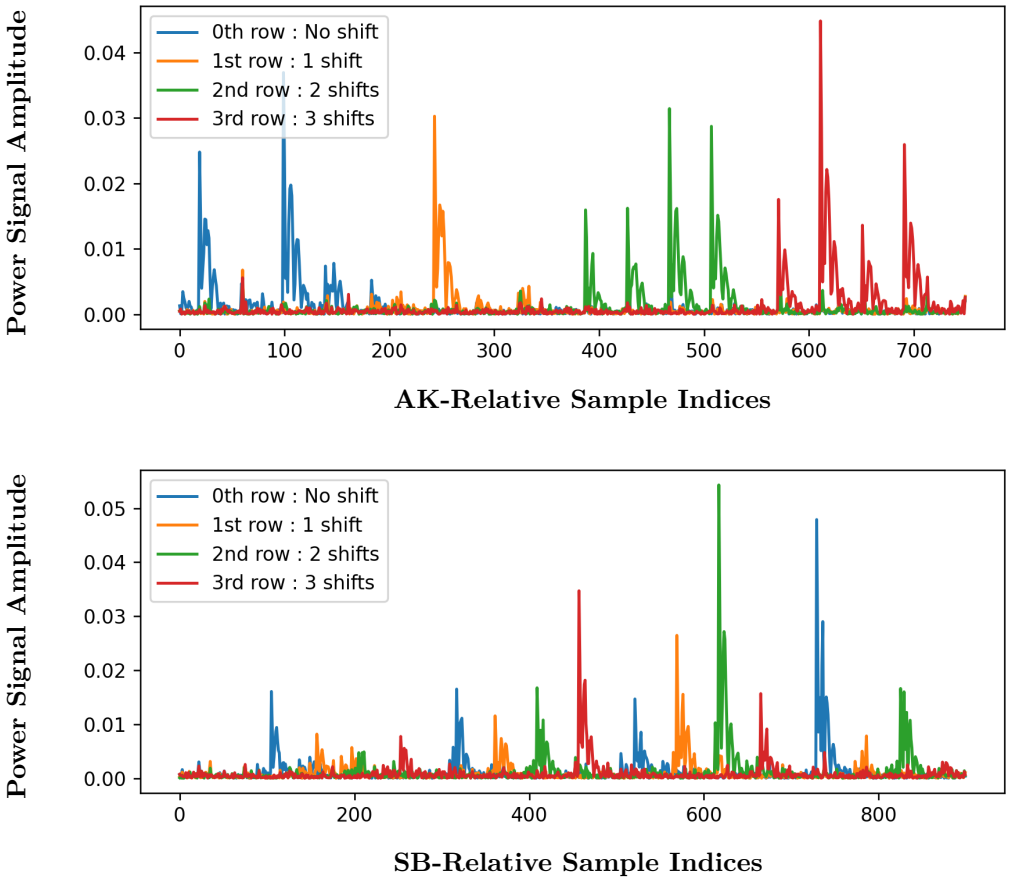


**Figure 5:** Four differentials power trace captures at AK(1) layer for top plot and at SB(1) layer for bottom plot. Implemented permutation Π is here the AES standard shift rows. Each differential power trace activates zeroth, first,second and third column in blue, orange, green and red respectively. We observe the column-major order of the AK layer instructions and the row-major order of the SB layer instructions.

$\forall i \in [0, 15]$. We focus arbitrarily on zeroth column, thus $u_0$, $u_1$, $u_2$ and $u_3$ are the bytes in the intermediate state that are mapped to the zeroth column after $PB(1)$. From previous step one can directly find the set of four byte indices activating one column, thus the set $\{u_0, u_1, u_2, u_3\}$ is known to the adversary. Moreover, let $d_0, d_1, d_2, d_3$ be equal to the four differentials at the first column after $PB(1)$,

$$\begin{aligned}
d_0 &:= b_{0,PB(1),p} + b_{0,PB(1),p'} = b_{\Pi(0),PB(1),p} + b_{Pi(0),PB(1),p'} \\
d_1 &:= b_{1,PB(1),p} + b_{1,PB(1),p'} = b_{\Pi(1),PB(1),p} + b_{\Pi(1),PB(1),p'} \\
d_2 &:= b_{2,PB(1),p} + b_{2,PB(1),p'} = b_{\Pi(2),PB(1),p} + b_{\Pi(2),PB(1),p'} \\
d_3 &:= b_{3,PB(1),p} + b_{3,PB(1),p'} = b_{\Pi(3),PB(1),p} + b_{\Pi(3),PB(1),p'}
\end{aligned} \tag{1}$$

We are now interested in the plaintext differentials diffusing to two active bytes after the $PB(1)$ layer. In order to do so the attack injects two differential plaintext bytes. With some probability this injection can lead to a diffusion of three active bytes after the $MC(1)$ layer. To compute number of bytes that are active after the $MC(1)$ layer, the attacker can simply count number of spikes in the differential power trace after $AK(1)$ or $SB(2)$. By collecting several relations from the differential power trace diffusing three active bytes

after the $MC(1)$ layer one can combine them to obtain a set of filter equations used to compute the coefficients of the matrix $M$. Algorithm 10 illustrates the procedure to obtain first filter equation and can be generalized to obtain the set of filter equations according to Table 1.

**Algorithm 10:** Find Hamming weight of variable $x_1$ in first filter equation

`// Set plaintext to be the same`

$p' \leftarrow p$

`// Set plaintext bytes in order to obtain` $d_0 = 255$

Set $p_{u_0}$ to be the byte s.t. $HW(b_{u_0,SB(1)}) = 0$ using modified Algorithm 7 to set HW after $SB(1)$.

Set $p_{u_1}$ to be the byte s.t. $HW(b_{u_1,SB(1)}) = 0$.

Set $p'_{u_0}$ to be the byte s.t. $HW(b_{u_1,SB(1)}) = 8$.

`// Find plaintext bytes in order to have a diffusion of three active bytes in zeroth column after` $MC(1)$

Set $p'_{u_1} \neq p_{u_1}$ to be the byte s.t. $\delta_{0,MC(1)} = 0$ using the fact that this should lead to no peak in the differential power trace $\bar{E}(b_{0,AK(1)}) - \bar{E}(b'_{0,AK(1)})$

`// As` $d_0 = 255, d_1 = x_1, d_2 = 0, d_3 = 0$

`// Obtained filter equation is:` $255a = x_1 b$

**return** $H(b'_{u_1,SB(1)}) = H(d_1) = H(x_1) = w_1$ using Algorithm 7.

**Table 1:** Inferred relations in function of the choice of the $d_i$ and the coefficients of the matrix $M$.

| Step | $HW(d_0)$ | $HW(d_1)$ | $HW(d_2)$ | $HW(d_3)$ | $\delta_{MC(1)}$ | Obtained relation |
|---|---|---|---|---|---|---|
| 1 | 8 | $HW(x_1)$ | 0 | 0 | $\delta_{0,MC(1)} = 0$ | $255a = x_1 b$ |
| 2 | $HW(x_2)$ | 8 | 0 | 0 | $\delta_{0,MC(1)} = 0$ | $255b = x_2 a$ |
| 3 | 8 | $HW(x_3)$ | 0 | 0 | $\delta_{1,MC(1)} = 0$ | $255d = x_3 a$ |
| 4 | $HW(x_4)$ | 8 | 0 | 0 | $\delta_{1,MC(1)} = 0$ | $255a = x_4 d$ |
| 5 | 8 | $HW(x_5)$ | 0 | 0 | $\delta_{2,MC(1)} = 0$ | $255c = x_5 d$ |
| 6 | $HW(x_6)$ | 8 | 0 | 0 | $\delta_{2,MC(1)} = 0$ | $255d = x_6 c$ |
| 7 | 8 | $HW(x_7)$ | 0 | 0 | $\delta_{3,MC(1)} = 0$ | $255b = x_7 c$ |
| 8 | $HW(x_8)$ | 8 | 0 | 0 | $\delta_{3,MC(1)} = 0$ | $255c = x_8 b$ |
| 9 | 8 | 0 | $HW(x_9)$ | 0 | $\delta_{0,MC(1)} = 0$ | $255a = x_9 c$ |
| 10 | $HW(x_{10})$ | 0 | 8 | 0 | $\delta_{0,MC(1)} = 0$ | $255c = x_{10} a$ |
| ... | | | | | | ... |

By combining the ten inferred relations in Table 1, one can show[1] that it reduces to the following filter equations,

$$\begin{aligned} x_1x_2 &= 255^2 & x_2x_8x_9 &= 255^3 \\ x_3x_4 &= 255^2 & x_1x_7x_{10} &= 255^3 \\ x_5x_6 &= 255^2 & x_2x_4x_6x_8 &= 255^4 \\ x_7x_8 &= 255^2 & x_1x_3x_5x_7 &= 255^4 \\ x_9x_{10} &= 255^2 \end{aligned}$$

From the filter equations above and the computed Hamming weights $H(x_i) = w_i$, there is always a valid solution for all $x_i$ whenever the indices $u_0, u_1, u_2$ and $u_3$ are correctly guessed. To recall, the attacker only knows the set $\{u_0, u_1, u_2, u_3\}$ but not its order. There are $4! = 24$ possible permutations of this set, and attacker must try them all and go through Algorithm 11 to find if its guess is correct up to shifting. Algorithm 11 outputs either no solution if the guess was wrong or output the same set of $2^8 - 1$ solutions for coefficients of $M$; $a, b, c$ and $d$ if attacker guess is correct up to shifting. Attacker obtains a more accurate description of the permutation $\Pi$ up to shifting among the permutations ending in same column by combining and repeating above stage for each state column. Indeed, for each column $c \in \{0, ..., 3\}$ the attacker finds $[u_{4*c}, u_{4*c+1}, u_{4*c+2}, u_{4*c+3}]$ array up to element shifting.

**Algorithm 11:** Find a set of possible solution for $a, b, c, d$ of size $2^8 - 1$

```
for n ← 0 to Threshold do
    Select a set of ten bytes b_1, ..., b_10 with b_i ∈ F_256 such that H(b_i) = w_i.
    if b_1, ..., b_10 is a solution to the filter equations then
        S = {}
        for a ← 0 to 255 do
            b ← 255^{-1} x_2 a
            c ← 255^{-1} x_9 a
            d ← 255^{-1} x_3 a
            S ← S + {[a, b, c, d]}
        return S
return None
```

To summarize, the attacker ends up with a set of $2^8 - 1$ solutions of matrix $M$ and with a description of the permutation $\Pi$ up to shifting among the permutations ending in same column which is sufficient to fully describe the $MC$ layer. Indeed, the permutation $\Pi$ up to shifting among the permutations ending in same column are all equivalent as they algebraically perform as the same $MC$ layer.

**Complexity:** The total complexity of this stage is about $4 \times 21 \times 26 \times \alpha \times 2^8$ measurement traces. $26 \times \alpha \times 2^8$ is the complexity to find the Hamming weights of the variable $x_1$ to $x_{10}$ via Algorithm 10. In a worst-case scenario, the attacker repeats previous step 21 times for each initial guess of the position bytes activating a same column (e.g. initial guess for $u0, u1, u2, u3$ activating zeroth column). Together with Algorithm 11, this finds a set of 255 solution for $M$. Finally, to obtain a description which is equivalent to $MC$ layer, attacker must repeat above four times for each activating column to find $\Pi$ up to shifting among the permutations ending in the same column.

### 3.4 Attack on Substitution Layer

Thanks to the previous section, attacker has 255 guesses for $M$. We take arbitrarily one of them. Using DPA the attack will find plaintext such that the convergence property [BBH+19] is verified. This convergence property is verified with probability $2^{-22}$ and happens when,

$$\begin{aligned}
b_{0,SB(1)} + b'_{0,SB(1)} &= T(p_0 + k_0) + T(p_0 + k_0 + d_0) = e\lambda \\
b_{1,SB(1)} + b'_{1,SB(1)} &= T(p_1 + k_1) + T(p_1 + k_1 + d_1) = h\lambda \\
b_{2,SB(1)} + b'_{2,SB(1)} &= T(p_2 + k_2) + T(p_2 + k_2 + d_2) = g\lambda \\
b_{3,SB(1)} + b'_{3,SB(1)} &= T(p_3 + k_3) + T(p_3 + k_3 + d_3) = f\lambda,
\end{aligned} \tag{2}$$

for any $\lambda$ in $\mathbb{F}_{256}^*$, and where $d_i$ were defined in Equation 1 and $e, h, g, f$ are the coefficients of the matrix $M^{-1}$ equal to,

$$M^{-1} = \begin{pmatrix} e & f & g & h \\ h & e & f & g \\ g & h & e & f \\ f & g & h & e \end{pmatrix}$$

One could simplify Equation 2 by setting the plaintext $p$ such that $b_{i,SB(1)} = 0$ for all $i$ in $\{0, 1, 2, 3\}$. This can be done using modified Algorithm 7 to set Hamming weight after $SB(1)$ to 0. Equation 2 is thus reduced to,

$$\begin{aligned}
T(p_0 + k_0 + d_0) &= e\lambda \\
T(p_1 + k_1 + d_1) &= h\lambda \\
T(p_2 + k_2 + d_2) &= g\lambda \\
T(p_3 + k_3 + d_3) &= f\lambda
\end{aligned} \tag{3}$$

Convergence property is found by an attacker when a differential active column in plaintext reduces into a single active byte in this same column after $MC(1)$. In average, about $2^{11.5}$ encryptions are required for the convergence to happen. More precisely, attacker counts number of spikes in the differential power trace after the $AK(1)$ or $SB(2)$ layers. Once found, use of Algorithm 7, allows the attacker to compute,

$$\begin{aligned}
HW(T(p_0 + k_0 + d_0)) &=: w_0 \\
HW(T(p_1 + k_1 + d_1)) &=: w_1 \\
HW(T(p_2 + k_2 + d_2)) &=: w_2 \\
HW(T(p_3 + k_3 + d_3)) &=: w_3
\end{aligned}$$

Combining the knowledge of $p_i, k_i$ and $d_i$ $\forall i \in \{0, 1, 2, 3\}$ and Equation 3 allows adversary to recover the S-box function $T$. To do so, Algorithm 12 finds in most cases four substitutions of the $T$ function where $e, f, g, h$ are computed from any of the 255 coefficients solutions found for $a, b, c, d$ in the previous section. The adversary repeats previous step for different differentials and for all the 255 possible solutions for $M$ (and thus for the different $e, f, g$ and $h$ coefficients).

**Complexity:** The simplification of Equation 2 has complexity $\alpha \times 4 \times 2^8$. Then running Algorithm 12 requires $\alpha \times \beta \times 2^{11.5}$ power traces and let the adversary find four substitutions among the 256 in $T$ table. $\beta$ is the inverse probability that a unique Hamming weight fingerprint in $L$ table is found. The total complexity is thus $\alpha \times 4 \times 2^8 + 256/4 \times \alpha \times \beta \times 2^{11.5}$.

**Algorithm 12:** Tries to find four substitutions of $T$

$L \leftarrow []$
**for** $\lambda \leftarrow 1$ **to** $255$ **do**
  $L[\lambda] \leftarrow [HW(e\lambda), HW(h\lambda), HW(g\lambda), HW(f\lambda)]$
// Return the four substitutions if an entry in $L$ equals $[w_0, w_1, w_2, w_3]$
**if** $[w_0, w_1, w_2, w_3] \in L$ **then**
  **return** $\{p_0 + k_0 + d_0 : e\lambda, p_1 + k_1 + d_1 : h\lambda, p_2 + k_2 + d_2 : g\lambda, p_3 + k_3 + d_3 : f\lambda\}$
  where $\lambda$ is s.t. $L[\lambda]$ equals $[w_0, w_1, w_2, w_3]$

# 4 SCARE Attack Applied to Cryptographically-Protected AES

This section extends the SCARE attack to protected AES-Like ciphers.

## 4.1 Key Recovery with Shuffled AK

### 4.1.1 Key Bytes Set Recovery with Shuffled AK

We consider AddRoundKey operations applied according to instructions shuffling among 16! possible instructions re-orderings. Recall that we denoted by $\Pi^{(k)}$ the associated AK random instructions permutation. In a protected (i.e. shuffling) setting we now observe sequentially $(E(b_{\Pi^{(k)}(i),AK(0)}), i \in \{0, ..., 15\})$ where $\Pi^{(k)}(\cdot)$ is unknown and randomized.

For example, if we set all plaintext bytes to `0x00` (i.e. $p_i =$ `0x00`, $i \in \{0, ..., 15\}$) and capture the entire power trace of its encryption at the $AK(0)$ layer we observe sequentially the $(E(k_{\Pi^{(k)}(i),AK(0)}))$, $i \in \{0, ..., 15\})$. Moreover, let us assume one of the key byte, say the $l$-th one, is also equal to `0x00`, thus $k_l :=$ `0x00` $= p_i$. From Section 2.2 we derive that the minimum $E(b_{\Pi^{(k)}(i),AK(0)})$ should correspond in average to $AK(0)$ encryption power trace of the byte xor-ed with $l$-th key byte (i.e. encryption of $p_l$ byte). Thus,

$$l = \underset{i \in \{0,...,15\}}{\arg\min}\ \bar{E}(b_{\Pi^{(k)}(i),AK(0)}) = \underset{i \in \{0,...,15\}}{\arg\min}\ \bar{E}(b_{i,AK(0)})$$

Figure 6 illustrates this example where the key was set to sixteen `0x00` bytes, and the plaintext was set to bytes different than the `0x00` byte except for the zeroth byte. Equivalently, $k_i =$ `0x00`, $\forall i\{0, ..., 15\}$, $p_0 =$ `0x00` and $p_i \neq$ `0x00` , $\forall i\{1, ..., 15\}$. We then observe that encryption of $p_0$ at $AK(0)$ layer was performed at seventh instruction (i.e. $\Pi^{(k)}(7) = 0$). Extending above observation, we see that if we iterate above all $t \in \mathbb{F}_{256}$ and set all plaintext bytes to $t$ and record each time

$$E_{min,p} = \min_{i \in \{0,...,15\}} \bar{E}(b_{i,AK(0)})$$

then the $t$'s corresponding to the 15-th lowest $E_{min,p}$ should each be equal to a key byte, assuming all the key bytes are different. Algorithm 13 finds in average the set of key bytes $\{k_0, ..., k_{15}\}$ when all the key bytes are different.

**Complexity**: Assuming a random uniform permutation, we denote for $AK(0)$ the random variable

$$B_{\Pi^{(k)}(i),AK(0)} := p_I + k_I$$

where $I \sim U(\{0, ..., 15\})$ is the random variable representing the random permutation choice for the $i$-th instruction (i.e. I $:= \Pi^{(k)}(i)$). We denote by $A$ the random variable counting the number of times $I = i$ (i.e. when we have trivial instructions permutation

**Algorithm 13:** Find a set containing the key bytes

```
   // Initialize an empty array of size 256
1  L ← [] // Iterate over all bytes
2  for i ← F_256 do
3      p ← t // Set all plaintext bytes to t
4      E_average ← 0
       // Capture β-power traces and compute average of the power trace
          minimum over each byte
       // β should be large enough to ensure stability
5      for j ← 0 to β do
           // Compute minimum average power trace over each key bytes
              indices and update average according
6          E_average ← E_average + min_{i∈{0,...,15}} Ē(b_{i,AK(0)})
       // Normalizing to compute average
7      L[t] ← E_average/β
   // Returns indices corresponding to the 16 lowest power trace
      averages in L
8  return (argsort L)[0 : 16]
```

for the $i$-th key byte). For each $t$ iterations we have $A \sim \mathcal{N}(\beta, \frac{1}{16})$ where $\beta$ is the number of power trace captures to obtain a stable algorithm. According to the $\alpha = 10$ parameter in Section 2.2 and choosing a threshold of 99.9% we ensure that for each iteration of $t$ we capture at least $\alpha$ trivial key permutation for a $i$-th key byte. We search for $Pr(\{A \leq \alpha - 1\}) = F_A(\alpha - 1) < 0.999$. With $\alpha = 10$, we compute $\beta = 356$. We thus have a final complexity of $\beta * 2^8$.

We will now talk about the case when some key bytes are equal. We observed heuristically that from a power trace $\bar{E}(b_{\Pi^{(k)}(i),AK(0)})$ we can estimate the Hamming weight of $b_{\Pi^{(k)}(i),AK(0)}$. Applying this to our case, from looking at the fifteenth lowest $E_{min,p}$ obtained at above paragraph, we can keep only the ones that correspond to $HW(b_{u,AK(0)}) = 0$ for any $u \in \{0, ..., 15\}$. We slightly modify Algorithm 13 in Algorithm 14 to compute (from one power trace capture) Hamming weight of state bytes after AK in permutation order i.e. we compute ordered $(HW(b_{\Pi^{(k)}(i),AK(0)}), i \in \{0, ..., 15\})$.

### 4.1.2 Key Bytes Permutation Recovery with Shuffled AK

In the previous section we successfully recovered the set of key bytes, we will now present a way to recover the original key permutation $K$.

To recover the first key byte $k_0$, we iteratively set the plaintext byte $p_0$ to one of the element of the key bytes set found in previous section (i.e. $p_0 \in \{k_0, ..., k_{15}\}$) while keeping others plaintext bytes values different from the key bytes set (i.e. $p_i \notin \{k_0, ..., k_{15}\}$ for all $i \in \{1, ..., 15\}$). At each iteration we compute $HW(b_{\Pi^{(k)}(i),AK(0)})$ for all $i \in \{0, ..., 15\}$ and observe if one those are equal to 0. If one of those is equal to 0 then it must be that $b_0 = p_0 + k_0 = 0$ as the other plaintext bytes are different than the key bytes. We must redo this attack 14 times to find $k_1$ up to $k_{14}$. We deduce $k_{15}$ from already found key bytes and the bytes set by proceeding by elimination. Algorithm 15 is a straightforward way to obtain the key permutation with a complexity of $16^2$.

**Worst-case complexity**: We can reduce Algorithm 15 complexity down to $(15 + 14 + ... + 1)\alpha = 120\alpha$. Indeed, in a first step we perform $\alpha$ power trace measurements of one encryption each time we change the value of $p_0$, and we only need to change $p_0$

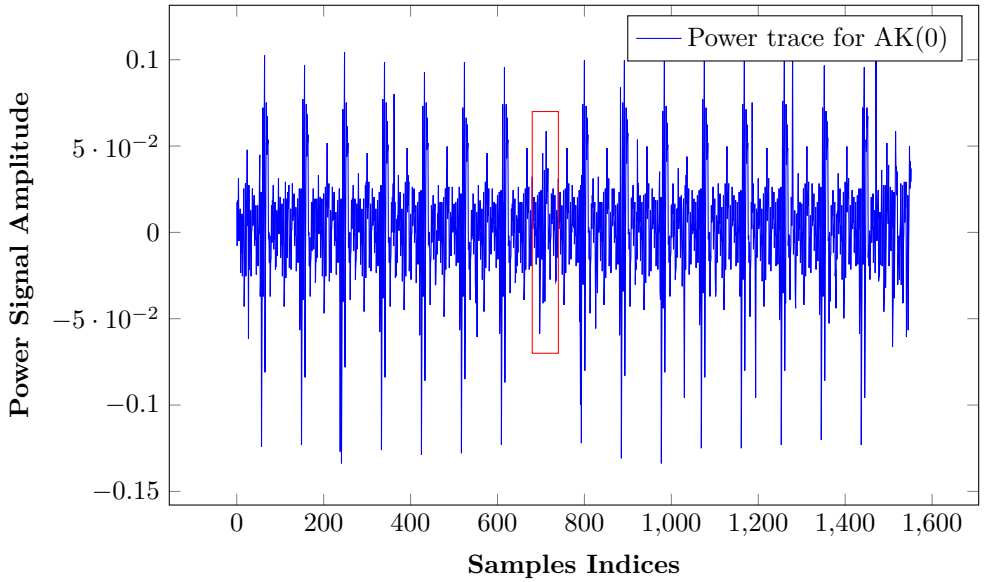


**Figure 6:** Power trace for AK(0). Red circled spike correspond to a minimal byte encryption power trace. Thus this byte is equal to its corresponding key byte.

value fifteen times (from $k_0$ to $k_{14}$), as if we did not observe the zero Hamming weight up to this, then it means that $p_0 = k_{15}$. We apply same the reasoning for the remaining plaintext bytes.

## 4.2 Substitution Recovery with Shuffled SB

### 4.2.1 Hamming Weight Recovery after SB Layer

In a similar way as shown in the previous section, we consider SubBytes operations applied according to instructions shuffling among 16! possible instructions reordering. As a recall we denoted by $\Pi^{(s)}$ the associated SB random instructions permutation. In a protected (i.e. shuffling) setting we now observe sequentially $(E(b_{\Pi^{(s)}(i),SB(1)}), i \in \{0, ..., 15\})$ where $\Pi^{(s)}(\cdot)$ is unknown and randomized.

In order to find the Hamming weight set of state bytes after the SB layer we modify Algorithm 14 in Algorithm 16. We thus recover the unordered set $\{HW(b_{i,SB(0))}), \forall i \in \{0, \ldots, 16\}\}$.

**Profiling** : We will now present in details the used profiling technique that has as properties to be inexpensive and accurate. In the same way as in the previous section, we observe heuristically that from a power trace $\bar{E}(b_{\Pi^{(s)}(i),SB(1)})$ we can estimate the Hamming weight of $b_{\Pi^{(s)}(i),SB(1)}$. Figure 8 illustrates this with an example. We clearly observe that power trace amplitude is correlated with the Hamming weight of the $b_{i,SB(1)} \in \{\texttt{0x00}, \texttt{0xF0}, \texttt{0xFF}\}$ for all $i \in \{0, ..., 15\}$.

Algorithm 17 presents how to perform profiling without knowing the substitution. As a reminder the key is known (from Section 4.1) and the S-box substitution is fixed and unknown but it must be that $T$ is one-to-one mapping in AES. We can thus observe (see Figure 7) that $|\{b \in \mathbb{F}_{256}, HW(T(b)) = i\}| = \binom{8}{i}$ which is the number of ways to choose $i$ ones in a 1-byte number. For example, if we set all bytes of plaintext to be equal to exact key bytes permutation $K$ then its power trace capture at layer $SB(1)$ must contains $\binom{8}{0} = 1$ full power trace (of 16 spikes) corresponding to a power trace capture of bytes s.t $HW(b_{\Pi^{(s)}(i),SB(1)}) = 0$. We can extend this to build our profile : we set

**Algorithm 14:** Find Hamming weight of state bytes after AK

```
// Initialize an empty array of size 16
H ← []
// Construct an array L of size 9 such that L[j] is average power
   trace when HW(b_{_,AK(0)}) = j
Procedure: Profiling
    ...
    return L
// Find HW(b_{Π^(k)(i),AK(0)}) for all i ∈ {0,...,15} given a power trace E
   and L
for i ← 0 to 15 do
    // Take the absolute value of the difference of each element of L
       by E_i. Find the minimal argument of this new array.
    H[i] ← argmin_{j∈{0,...,8}} |L[j] − E_i|
return H
```

**Algorithm 15:** Find key permutation

```
// Key bytes permutation solution
K ← []
// Key bytes set
K_set ← {k_i}
// Set plaintext bytes to bytes not in K_set
p ← [p_i, with p_i ∉ K_set]
// Set i-th pt byte to an element of K_set and compute all Hamming
   weights HW(b_{,AK(0)}) using Algorithm 14, if one is equal to 0 then
   we found i-th key byte
for i ← 0 to 15 do
    for k ∈ K_set do
        p_i ← k
        H ← Algorithm 14
return K
```

all bytes of plaintext to be equal to $K$ we then iterate over $t \in \mathbb{F}_{256}$ and add to each plaintext byte $t$ and capture a power trace at layer $SB(1)$. The intermediate state bytes are represented in Figure 11. We then observe $\binom{8}{i}$ full power traces $E(b_{\Pi^{(s)}(j),SB(1)})$ such that $HW(b_{\Pi^{(s)}(j),SB(1)}) = i$ where $j$ is spike index.

The profiling phase consists in recording each spike trace while iterating over the plaintext bytes in order to obtain better accuracy without increasing complexity. For example for the first spike, thus corresponding to S-box substitution of the $b_{\Pi^{(s)}(0),SB(1)}$ state byte, we plot in Figure 9 the 256 recorded power traces of the first spike while iterating over $t$ (i.e. $p_i = k_i + t$ for $t \in \mathbb{F}_{256}$ and all $i \in \{0, ..., 15\}$). Power trace amplitude order is represented as a linear color gradient. For the sake of completeness Figure 10 plots the 256 recorded power traces of the first spike while iterating over $t$ but each trace is now colored with the true Hamming weight value of $b_{\Pi^{(s)}(0),SB(1)}$. To recap we created a profiling per spike that allows from observation of one power trace capture to estimate the set $\{HW(b_{\Pi^{(s)}(i),SB(1)}) \ \forall i \in \{0, ..., 15\}\}$. This technique was experimentally verified to have a high accuracy on the estimated Hamming weight set.

**Algorithm 16:** Find Hamming weight set of state bytes after SB

```
   // Initialize an empty array of size 16
1  H ← []
   // Construct an array L of size 9 such that L[j] is average power
      trace when HW(b_{,SB(1)}) = j
2  Procedure: Profiling
3      ...
4      return L
   // Find HW(b_{Π^(k)(i),SB(1)}) for all i ∈ {0,...,15} given a power trace E
      and L
5  for i ← 0 to 15 do
       // Take the absolute value of the difference of each element of L
          by E_i. Find the minimal argument of this new array.
6      H[i] ← arg min_{j∈{0,...,8}} |L[j] − E_i|
7  return H
```

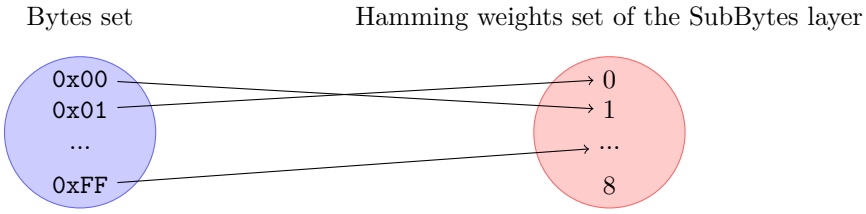


**Figure 7:** Mapping diagram of $b \mapsto HW(T(b))$ from bytes set $\mathbb{F}_{256}$ to Hamming weights set $\{0, ..., 8\}$. As $T$ is a one-to-one function then $|\{b \in \mathbb{F}_{256},\ HW(T(b)) = i\}| = \binom{8}{i}$

**Algorithm 17:** Profiling procedure : Compute a profile per spike

```
   // Initialize an array containing 16 arrays of empty size
1  L_profile ← [[] * 16]
   // Fill L_profile s.t L[j] is an array of size 9 containing in
      ascending order an estimate of the average power trace of
      HW(b_{,SB(1)}) for spike in position j. In other words,
      L_profile[j][i] = E{E, with HW(b_{Π^(s)(j),SB(1)}) = i}
2  for t ← 0 to 256 do
       // Set plaintext bytes to key bytes (computed in Section 4.1)
          plus t
3      p ← [k_i + t, i ∈ {0,...,15}]
       // Record a power trace E of SB(1) and append it to L_profile
4      for i ← 0 to 15 do
5          L_profile[i][t] = E(b_{Π^(s)(i),SB(1)})
6  for i ← 0 to 15 do
7      L_profile[i] = sort(L_profile[i])
8  return L_profile
```

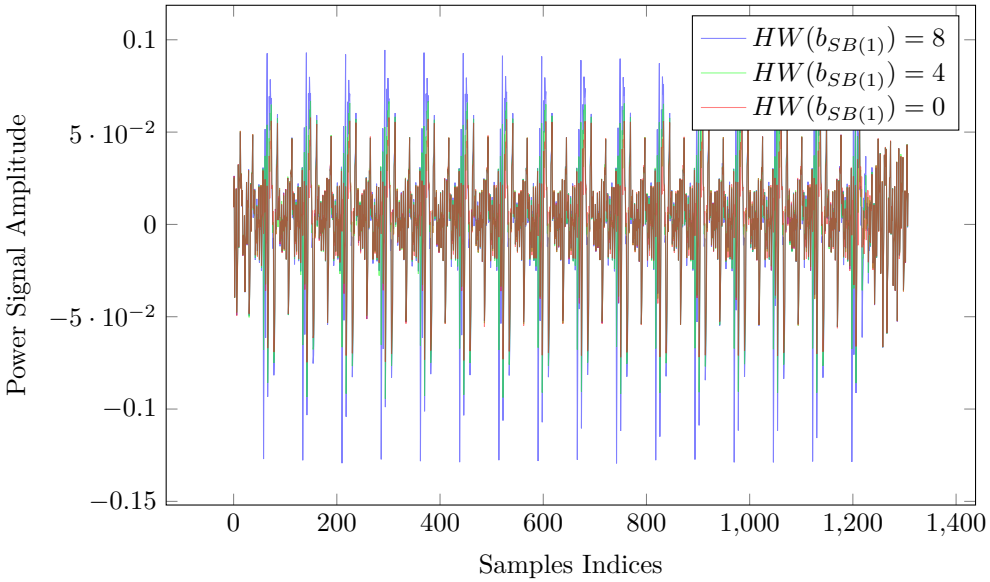


**Figure 8:** Power trace after SB(1) layer. For blue, green & red plots the plaintext bytes are all s.t $b_{SB(1)}$ equal `0xFF`, `0xF0` & `0x00` respectively thus of Hamming weight of 8, 4 & 0 respectively. From power trace spikes shape we can thus estimate Hamming weights of each $b_{\Pi^{(s)}(i),SB(1)}$. For example, by analyzing first spike shape we can compute $\hat{HW}(b_{\Pi^{(s)}(0),SB(1)})$.

### 4.2.2 Set the Hamming Weight of a Specific State Byte after SB Layer

As before, the algorithm recovering Hamming weight of the state bytes after $SB(1)$ layer can be extended via DPA to enforce any state bytes after $SB(1)$ to have a specified Hamming weight $h$.

In a first approach to set $HW(b_{i,SB(1)}) = 0$, we could conduct a DPA while keeping other plaintext bytes such that $HW(b_{j,SB(1)}) \neq 0 \ \forall j \neq i$. We can ensure this by using Algorithm 16 and thus by observing that the power trace has only one unique small spike representing the fact that byte $b_{i,SB(1)}$ is the only byte that has a null Hamming weight.

More generally to set $HW(b_{i,SB(1)}) = h$ to any Hamming weight value $h$, we simply need to look at the profiling we made. Indeed, if during profiling one record the byte $t$ associated to each power trace, then from a given Hamming weight $h$ one can look at the power trace amplitude position and infer all the bytes $t$ that gives $HW(T(t)) = h$. As an example, by looking at the second up to the ninth lowest power traces in Figure 9, we find the power traces that each correspond to a different byte $t$ verifying $HW(T(t)) = 1$.

### 4.2.3 Find plaintext bytes activating the same column

The last ingredient to apply the SCARE attack is the way to find the state bytes position activating a same column after $PB(1)$ layer. Thanks to Section 4.1 we recovered the key, and we can always recover all $b_{i,AK(0)} = p_i + k_i, \forall i \in \{0, \ldots, 16\}$. Again, thanks to this Section 4.2, we are able to compute Hamming weight set after SB; $\{HW(b_{i,SB(0))}), \forall i \in \{0, \ldots, 16\}\}$.

More precisely, are even able to compute any specific $HW(b_{i,SB(0))}), \forall i \in \{0, \ldots, 16\}$. Here is a way to do it:

- We recall the pipeline presented in Figure 11. We set all plaintext bytes to be equal to their corresponding key byte plus a value $t$ (i.e. $p_i = k_i + t$) while we iterate $t$

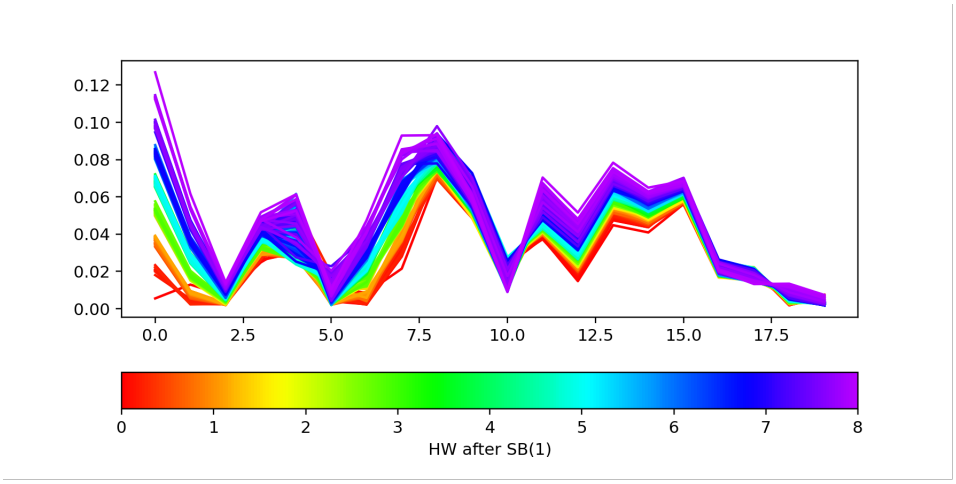


**Figure 9:** First power trace spike recorded after $SB(1)$ layer and for all possible constant plaintext. Power traces are color-sorted by **estimated** Hamming weight of the state byte corresponding to this spike after $SB(1)$

over all byte values, and we compute the Hamming weight value after SB (same for all bytes). For example with $t = 0$, we compute $HW(T(0))$ as all state bytes after AK are equal to $t = 0$ and all state bytes after SB are equal to $T(0)$). After the iteration over $t$ we found the $t =: j$ such that all state bytes after SB have Hamming weight 0 (i.e. all state bytes after SB are equal to 0). Thus $j$ is the byte value that is substitute by SB to 0.

- To compute $HW(b_{i,SB(0)})$, we set all plaintext bytes, except the $i$-th one, to their corresponding key byte plus $j$. And we set $p_i$ to any value different than $k_i + j$. By computing the set $\{HW(b_{u,SB(0)}), \forall u \in \{0, \ldots, 16\}\}$ using Algorithm 16, we recover $HW(b_{i,SB(0)})$ as the only value in this set that is not equal to 0.

If we set all the plaintext bytes, except the $i$-th one, to their corresponding key byte plus $j$ then $b_{u,SB(1)} = 0 \ \forall u \neq i$ as discussed above. Thus after the permutation, only one state byte is not equal to zero (the one where $p_{i,SB(1)}$ has been mapped) and the other state bytes after PB are equal to zero. After $MC(1)$, only four state bytes (in a same column) are not equal to zero (the ones in same column where $p_{i,SB(1)}$ has been mapped) and other state bytes after MC are equal to zero. We then compute the set $S := \{HW(b_{u,SB(2)}) \ \forall u \in \{0, ..., 15\}\}$ while changing the position of the byte $p_i$ (i.e we iterate over $i \in \{0, ..., 15\}$). Then we will observe for the bytes activating the same column that their associated sets $S$ have more than 12 values in common. In the opposite way, for the bytes **not** activating the same column their associated sets $S$ have high probability to have less than 12 values in common. That is how the attack goes to find the bytes activating the same column after $MC(1)$ layer. To sum up, Figure 13 and Figure 12 are respectively examples of scenario where the $p_0$ and the $p_1$ plaintext bytes are activating or not activating the same column after $PB(1)$ layer.

By assuming that the key or plaintext ($p_i$) are uniformly distributed we then have an empiric probability around 0.9[2] that the attack will work (i.e. find the 4 bytes activating the same column), in other cases one will find more than four (i.e. 8, 12, or 16) bytes activating the same column and the attacker simply needs to repeat this step with a different plaintext $p_i = t$.

[2]See `compute-prob-attack-find-activating-col.py` in my GitHub repository [Ren22]

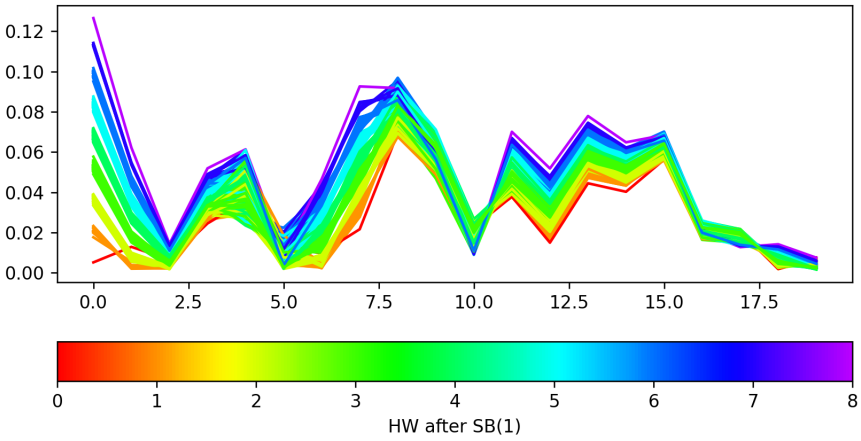


**Figure 10:** First power trace spike recorded after $SB(1)$ layer and for all possible constant plaintext. Power traces are color-sorted by **true** Hamming weight of the state byte corresponding to this spike after $SB(1)$

$$\begin{bmatrix} k_0+t & k_4+t & k_8+t & k_{12}+t \\ k_1+t & k_5+t & k_9+t & k_{13}+t \\ k_2+t & k_6+t & k_{10}+t & k_{14}+t \\ k_3+t & k_7+t & k_{11}+t & k_{15}+t \end{bmatrix} \xrightarrow{\text{AK}(0)} \begin{bmatrix} t & t & t & t \\ t & t & t & t \\ t & t & t & t \\ t & t & t & t \end{bmatrix} \xrightarrow{\text{SB}(1)}$$

$$\begin{bmatrix} T(t) & T(t) & T(t) & T(t) \\ T(t) & T(t) & T(t) & T(t) \\ T(t) & T(t) & T(t) & T(t) \\ T(t) & T(t) & T(t) & T(t) \end{bmatrix}$$

**Figure 11:** Operations pipeline when performing the profiling

## 4.3 Reverse the SB, PB and MC Layers

We now have all the ingredients to conduct the SCARE attack by following back Sections 3.2, 3.3 & 3.4. More precisely,

- We already reversed the AK layer in Section 4.1.
- We (partially)-reversed the PB layer in Section 4.2.3 by finding the permutation $\Pi$ up to row-permutation and up to column-permutation. More precisely we deduced the bytes activating a same column. As a note, we don't know the exact column activated contrary to the unprotected AES case. But the SCARE attack from Caforio et al. are not making use of this knowledge. There are 16! possibilities for the original random and hidden permutation $\Pi$. Attack can thus boiled down this to $4! * 4 * 4!$ (whereas in SCARE paper they reduced it down to $4 * 4!$ possibilities for the unprotected AES ciphers)
- We can reverse MC layer by making use of the Section 3.3 and setting during the attack a suitable plaintext such that the DPA can still be conducted. This can be achieved by using the set trick that we used to find bytes activating same column.

We note that we are still doing the attack over plaintext bytes activating the same column but knowledge of the column index is irrelevant.

- We can reverse the SB layer by making use of the Section 3.4 as we can set the Hamming weight of a specific state byte after the SB layer, and we can detect a convergence by also using a suitable plaintext.

Situation 1 : $i = 0$ and $p_0$ activates column 2.

$$\begin{bmatrix} k_0+t & k_4+j & k_8+j & k_{12}+j \\ k_1+j & k_5+j & k_9+j & k_{13}+j \\ k_2+j & k_6+j & k_{10}+j & k_{14}+j \\ k_3+j & k_7+j & k_{11}+j & k_{15}+j \end{bmatrix} \xrightarrow{\text{AK(0)}} \begin{bmatrix} t & j & j & j \\ j & j & j & j \\ j & j & j & j \\ j & j & j & j \end{bmatrix} \xrightarrow{\text{SB(1)}}$$

$$\begin{bmatrix} T(t) & 0 & 0 & 0 \\ T(j)=0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{PB(1)}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & T(t) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{MC(1)}}$$

$$\begin{bmatrix} 0 & 0 & b*T(t) & 0 \\ 0 & 0 & a*T(t) & 0 \\ 0 & 0 & d*T(t) & 0 \\ 0 & 0 & c*T(t) & 0 \end{bmatrix} \xrightarrow{\text{AK(1), SB(2)}} \begin{bmatrix} T(k_0) & T(k_4) & T(k_8+b*T(t)) & T(k_{12}) \\ T(k_1) & T(k_5) & T(k_9+a*T(t)) & T(k_{13}) \\ T(k_2) & T(k_6) & T(k_{10}+d*T(t)) & T(k_{14}) \\ T(k_3) & T(k_7) & T(k_{11}+c*T(t)) & T(k_{15}) \end{bmatrix}$$

Situation 2 : $i = 1$ and $p_0$ activates column 0.

$$\begin{bmatrix} k_0+j & k_4+j & k_8+j & k_{12}+j \\ k_1+t & k_5+j & k_9+j & k_{13}+j \\ k_2+j & k_6+j & k_{10}+j & k_{14}+j \\ k_3+j & k_7+j & k_{11}+j & k_{15}+j \end{bmatrix} \xrightarrow{\text{AK(0)}} \begin{bmatrix} j & j & j & j \\ t & j & j & j \\ j & j & j & j \\ j & j & j & j \end{bmatrix} \xrightarrow{\text{SB(1)}}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ T(t) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{PB(1)}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ T(t) & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{MC(1)}}$$

$$\begin{bmatrix} d*T(t) & 0 & 0 & 0 \\ c*T(t) & 0 & 0 & 0 \\ b*T(t) & 0 & 0 & 0 \\ a*T(t) & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{AK(1), SB(2)}} \begin{bmatrix} T(k_0+d*T(t)) & T(k_4) & T(k_8) & T(k_{12}) \\ T(k_1+c*T(t)) & T(k_5) & T(k_9) & T(k_{13}) \\ T(k_2+b*T(t)) & T(k_6) & T(k_{10}) & T(k_{14}) \\ T(k_3+a*T(t)) & T(k_7) & T(k_{11}) & T(k_{15}) \end{bmatrix}$$

**Figure 12:** A scenario where the bytes $p_0$ and $p_1$ are not activating the same column. The set $S$ for situation 1 and the set $S'$ for situation 2 have high probability to have strictly less than 12 elements in common.

Situation 1 : $i = 0$ and $p_0$ activates column 2.

$$\begin{bmatrix} k_0+t & k_4+j & k_8+j & k_{12}+j \\ k_1+j & k_5+j & k_9+j & k_{13}+j \\ k_2+j & k_6+j & k_{10}+j & k_{14}+j \\ k_3+j & k_7+j & k_{11}+j & k_{15}+j \end{bmatrix} \xrightarrow{\text{AK(0)}} \begin{bmatrix} t & j & j & j \\ j & j & j & j \\ j & j & j & j \\ j & j & j & j \end{bmatrix} \xrightarrow{\text{SB(1)}}$$

$$\begin{bmatrix} T(t) & 0 & 0 & 0 \\ T(j)=0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{PB(1)}} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & T(t) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{MC(1)}}$$

$$\begin{bmatrix} 0 & 0 & b*T(t) & 0 \\ 0 & 0 & a*T(t) & 0 \\ 0 & 0 & d*T(t) & 0 \\ 0 & 0 & c*T(t) & 0 \end{bmatrix} \xrightarrow{\text{AK(1), SB(2)}} \begin{bmatrix} T(k_0) & T(k_4) & T(k_8+b*T(t)) & T(k_{12}) \\ T(k_1) & T(k_5) & T(k_9+a*T(t)) & T(k_{13}) \\ T(k_2) & T(k_6) & T(k_{10}+d*T(t)) & T(k_{14}) \\ T(k_3) & T(k_7) & T(k_{11}+c*T(t)) & T(k_{15}) \end{bmatrix}$$

Situation 2 : $i = 1$ and $p_0$ activates column 2.

$$\begin{bmatrix} k_0+j & k_4+j & k_8+j & k_{12}+j \\ k_1+t & k_5+j & k_9+j & k_{13}+j \\ k_2+j & k_6+j & k_{10}+j & k_{14}+j \\ k_3+j & k_7+j & k_{11}+j & k_{15}+j \end{bmatrix} \xrightarrow{\text{AK(0)}} \begin{bmatrix} j & j & j & j \\ t & j & j & j \\ j & j & j & j \\ j & j & j & j \end{bmatrix} \xrightarrow{\text{SB(1)}}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ T(t) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{PB(1)}} \begin{bmatrix} 0 & 0 & T(t) & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \xrightarrow{\text{MC(1)}}$$

$$\begin{bmatrix} 0 & 0 & a*T(t) & 0 \\ 0 & 0 & d*T(t) & 0 \\ 0 & 0 & c*T(t) & 0 \\ 0 & 0 & b*T(t) & 0 \end{bmatrix} \xrightarrow{\text{AK(1), SB(2)}} \begin{bmatrix} T(k_0) & T(k_4) & T(k_8+a*T(t)) & T(k_{12}) \\ T(k_1) & T(k_5) & T(k_9+d*T(t)) & T(k_{13}) \\ T(k_2) & T(k_6) & T(k_{10}+c*T(t)) & T(k_{14}) \\ T(k_3) & T(k_7) & T(k_{11}+b*T(t)) & T(k_{15}) \end{bmatrix}$$

**Figure 13:** A scenario where the bytes $p_0$ and $p_1$ are activating the same column. The set $S$ for situation 1 and the set $S'$ for situation 2 have high probability to have more or equal than 12 elements in common.

# 5 Conclusion & Future work

In a first stage, this project introduced Side-Channel Attacks for AES-like ciphers implemented on a 32-bits microcontroller. The work from Caforio et al. [CBB21] were then reviewed in order to conduct again the full reverse-engineering of AES-128 like ciphers structure. Thanks to a new inexpensive and simple profiling technique, we developed a reverse-engineering attack on a protected AES cipher implementing instructions shuffling at the AddRoundKey and SubBytes layers. As this SCARE attack focuses on the $AK$ and $SB$ layers, this attack could also apply on AES ciphers implementing instructions shuffling at MixColumn and Permutation layers.

In the last decade, much progress have been done in the field of Side Channel Attack and the need to practically evaluate the effectiveness of existing countermeasures is growing. This work suggests much future work to be done and the possibility to reverse-engineer protected AES-like ciphers implementing more protections like random masking. A similar SCARE procedure may also be extended beyond AES ciphers or ciphers implementing software-level changes (like the T-table implementations [MWK17], the ANSSI AES [LoES18] implementation or the x86 implementations other than 32-bit ARM) or implementing more complex hidden functions. Finally, it looks like machine learning methods could easily be integrated in some profiling or power trace analysis techniques. Indeed, goals of such techniques are typically to learn from and estimate the side-channel leakages which could then represent or generate large amount of data leakage, an important criterion in machine learning.

## References


[AG] Mehdi-Laurent Akkar and C. Giraud. An Implementation of DES and AES, Secure against Some Attacks. pages 309–318.

[AMN+11] Antoine Amarilli, Sascha Müller, David Naccache, Daniel Page, Pablo Rauzy, and Michael Tunstall. Can code polymorphism limit information leakage? In Claudio A. Ardagna and Jianying Zhou, editors, *Information Security Theory and Practice. Security and Privacy of Mobile Devices in Wireless Communication*, pages 1–21, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[BBH+19] Shivam Bhasin, Jakub Breier, Xiaolu Hou, Dirmanto Jap, Romain Poussier, and Siang Meng Sim. Sitm: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):95–122, Nov. 2019.

[BJB18] Jakub Breier, Dirmanto Jap, and Shivam Bhasin. Scadpa: Side-channel assisted differential-plaintext attack on bit permutation based ciphers. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1129–1134, 2018.

[CB08] D. Canright and Lejla Batina. A very compact "perfectly masked" s-box for aes. In Steven M. Bellovin, Rosario Gennaro, Angelos Keromytis, and Moti Yung, editors, *Applied Cryptography and Network Security*, pages 446–459, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[CBB21] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Complete practical side-channel-assisted reverse engineering of aes-like ciphers. Cryptology ePrint Archive, Paper 2021/1252, 2021. https://eprint.iacr.org/2021/1252.

[CIW13] Christophe Clavier, Quentin Isorez, and Antoine Wurcker. Complete scare of aes-like block ciphers by chosen plaintext collision power analysis. In Goutam Paul and Serge Vaudenay, editors, *Progress in Cryptology – INDOCRYPT 2013*, pages 116–135, Cham, 2013. Springer International Publishing.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security*, pages 239–252, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[Inc22] NewAE Technology Inc. Chipwhisperer. https://github.com/newaetech/chipwhisperer, 2022.

[JB20] Dirmanto Jap and Shivam Bhasin. Practical reverse engineering of secret sboxes by side-channel analysis. pages 1–5, 10 2020.

[KHL11] HeeSeok Kim, Seokhie Hong, and Jongin Lim. A fast and provably secure higher-order masking of aes s-box. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 95–107, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[LoES18] ANSSI Laboratory of Embedded Security. Secure AES 128 on ATMEGA8515. https://github.com/ANSSI-FR/secAES-ATmega8515, 2018.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. 01 2007.

[MWK17] Heiko Mantel, Alexandra Weber, and Boris Köpf. A systematic study of cache side channels across aes implementations. In Eric Bodden, Mathias Payer, and Elias Athanasopoulos, editors, *Engineering Secure Software and Systems*, pages 213–230, Cham, 2017. Springer International Publishing.

[Nov03] R. Novak. Side-channel attack on substitution blocks. In J. Zhou, M. Yung, and Y. Han, editors, *ACNS 2003. LNCS*, volume 2846, page 307–318. Springer, Heidelberg, 2003.

[Ren22] Gaiëtan Renault. Apply Practical Side-Channel-Assisted Reverse Engineering attack to Protected AES Cipher. https://github.com/grennault/MSc_Semester_Project_SCARE_Frontier, 2022.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 171–188, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[RR13] Matthieu Rivain and Thomas Roche. Scare of secret ciphers with spn structures. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013*, pages 526–544, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 740–757. Springer, 2012.